University of California
Santa Cruz

Distributed Assignment of Codes in Multihop Radio Networks

A thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Computer Science

by

Jyoti Raju

June 1998

The thesis of Jyoti Raju is approved:

Prof. J.J. Garcia-Luna-Aceves

Prof. Anujan Varma

Prof. Darrell D.E. Long

Dean of Graduate Studies and Research

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
JUN 1998		**00-06-1998 to 00-06-1998**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Distributed Assignment of Codes in Multihop Radio Networks	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Santa Cruz,Department of Computer Engineering,Santa Cruz,CA,95064**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**59**	

This thesis is dedicated to my grandfather – who taught me to enjoy learning and to keep God as a constant companion.

Contents

List of Figures

Distributed Assignment of Codes in Multihop Radio Networks

Jyoti Raju

ABSTRACT

Code assignment is necessary for the proper functioning of an ad-hoc CDMA network. Due to the irregular topology of ad-hoc networks, an optimal and distributed solution for the code-assignment problem is *NP-Complete*.

This thesis presents a distributed greedy algorithm for assigning codes in a dynamic, multihop wireless CDMA radio network. The same algorithm can be used to assign channels in a multichannel CSMA/CA network. The algorithm does not require any form of synchronization and is completely distributed. It can be used for both the transmitter oriented and receiver oriented code assignment.

The algorithm is proven to be correct and its complexity is analyzed. The algorithm has been implemented to assign channels in a multi-channel CSMA/CA network. The results from the implementation are presented here and it is shown that the algorithm is scalable and works correctly in mobile environments.

Keywords: CDMA, Dynamic Channel Assignment, Code Assignment

Acknowledgements

I am especially grateful to my advisor, J.J. Garcia-Luna-Aceves for his invaluable guidance and support. His sense of humour and patience makes working with him really worthwhile.

I would like to thank Prof. Anujan Varma and Prof. Darrell D. E. Long for being part of my committee and for teaching great classes. Thanks are also due to Chane and Mumu, my CPT gurus for answering the dumbest of questions.

A big thank you to all of CCRG and the folks up in AS 350. This work would not have been as much fun without you all. Special thanks to Abby and Chris for keeping my life on "schedule" and to Shree and Pratibha for always being there.

Above all, I'd like to thank my mom, my dad and my sister. I am highly indebted to them for the "reality checks" they subject me to from time to time.

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under grant DAAB07-95-C-D157.

1. Introduction

Medium access control (MAC) protocols are basic building blocks of packet radio networks. Because these protocols control the media directly, they need to take into consideration the low bandwidth and high interference characteristics of the wireless environment. There have been a wide variety of MAC protocols for the wireless environment; these protocols can be classified into those that use a single channel for transmission and those that use multiple channels.

There are a wide variety of single-channel MAC protocols. These include schemes, based on fixed channel assignment like FDMA and TDMA [BG94]. In FDMA the share of the channel consists of a narrow frequency band and in TDMA the share consists of a time slot. FDMA and TDMA are ideally suited for a low peak to average ratio traffic; however, if the traffic is bursty, these protocols result in low bandwidth usage. The advantage of using these algorithms is that they provide delay bounds for isochronous traffic.

At the other end of the spectrum in single-channel MAC protocols are random-access protocols like (ALOHA) [Abr70], Carrier Sense Multiple Access (CSMA) [TK75a], Multiple Access Collision Avoidance (MACA) [Kar90], MACA for Wireless (MACAW) [VBZ94] and Floor Acquisition Multiple Access (FAMA) [FGLA97]. These protocols require every station to contend for the channel if there is traffic to send. A key advantage of these contention-based MAC protocols is that they require very little synchronization and no centralized control. However, their performance degrades as traffic increases due to higher rates of collisions, which is one of the reasons random-access protocols cannot guarantee a bounded maximum delay for data traffic.

One way to increase the throughput of a random access protocol is to have stations resolving collisions on a control channel while sending data in separate data channels, ideally one channel for every node or station. This prevents neighboring stations from interfering with one another while transmitting data packets and also increases throughput. However, it is not always possible to have as many data channels as there are nodes in the network.

This situation creates the channel-reuse problem with the constraint that stations two hops away from each other cannot share the same data channel in order to avoid interference. This thesis presents a distributed algorithm that solves the channel reuse problem. We call this algorithm the Code Assignment and Dissemination Algorithm (CADA). The algorithm dynamically assigns data channels to stations in a way that no two-hop neighbors have the same data channel and therefore there is no interference while transmitting a data packet. CADA uses the MAC protocol and the routing protocol of the network to acquire and disseminate information respectively.

Another application of the channel assignment algorithm is to distribute codes in a code division multiple access (CDMA) network. CDMA networks use spread spectrum techniques to transfer data. Using CDMA in packet-radio networks permits multiple stations within range of the same receivers to transmit concurrently and without interference as long as the stations do not have the same code. Several multi-access protocols have been proposed and commercial systems have been deployed that take advantage of CDMA [Ric]. An important design consideration in a multihop packet-radio network using CDMA is the assignment of transmission codes to network nodes. CDMA has mainly been used in cellular networks in which a base station takes on the task of assigning codes. In an ad-hoc network, this task must be done distributedly. The situation here is similar to the one in the multi-channel random-access case. The number of transmission codes is smaller than the number of nodes, and senders and receivers must agree on which transmission code to use in a way that avoids interference as much as possible. This means that stations two hops away from each other need to use different codes [Hu93]. Both the channel-assignment and code-assignment problems can be mapped onto the two-hop node coloring problem of a graph.

Chapter 2 presents a brief survey of random-access protocols and previous work done in the area of dynamic channel or code allocation. Chapter 3 describes our new algorithm for distributed assignment of codes or channels. In Chapter 4 we prove that the algorithm guarantees correct channel/code assignments after convergence, provided that there are enough channels/codes available. Chapter 5 addresses the algorithm's complexity. Chapter

6 presents the results of simulation experiments. We aimed at analyzing the increase in throughput due to correct channel/code assignment, the scalability of the algorithm and its adaptivity to mobility. Finally, Chapter 7 presents our conclusions and ideas for future work.

2. Background

2.1 Random-Access Protocols

The ALOHA protocol [Abr70] was one of the first random-access protocols to be used in wireless communication. This protocol allows stations to transmit any time they desire. If within some timeout no acknowledgment is received for the transmitted packet, then the station assumes that a collision occurred and the packet needs to be retransmitted. An improvement over pure ALOHA is to synchronize the channel by slotting time into segments whose duration is exactly equal to the time for transmission of a single packet. This is know as slotted ALOHA [Abr73, KL73]. Since each station starts packets only at the beginning of a slot, when packets conflict, it is ensured that they will overlap completely rather than partially, providing an increase in channel efficiency.

CSMA [TK75a] uses carrier sensing to avoid collisions by listening to the carrier due to another stations transmission. An important assumption used here is that the propagation delay on the channel is much smaller than a packet transmission time. Since, the vulnerability period is reduced from the packet transmission time to the propagation time, CSMA represents a substantial improvement over ALOHA. The throughput of CSMA protocols is very good as long as there are no "hidden terminals". This means that there are stations which cannot hear each other and consequently might send packets at the same time to a receiver causing the packets to collide. In the presence of hidden terminals performance of CSMA degrades to that of ALOHA.

One of the first solutions to the hidden terminal problem was the BTMA [TK75b] protocol. This protocol required each wireless terminal to be within line-of-sight of a base station. The base station transmits a busy tone on a channel different from the data channel. Thus, every terminal has to listen to the channel on which the busy tone is sent, before transmitting. The main disadvantages of BTMA are that it requires an extra channel and that it requires a receiver to transmit a busy tone while receiving data.

The next phase of wireless protocols was those using CSMA/CA techniques. Here a station that has a packet to send, sends a request to send (RTS) over the channel. If the receiver gets the packet correctly, it sends a clear to send (CTS). The CTS tells the sender when to send the data. The advantage over CSMA protocols is that the data packet goes in clear. Only the RTS's face contention. There have been many proposals of CSMA/CA to date. The basic method was first proposed in SRMA [TK76]. MACA [Kar90] was a CSMA/CA protocol that did not use carrier sensing, just like the ALOHA-based variation of SRMA. MACAW [VBZ94] is a variation of MACA with better retransmission strategies. However, both MACA and MACAW suffer from hidden-terminal interference, as proven in [FGLA97].

FAMA-NCS [FGLA97] solves the hidden-terminal problem in multihop networks. This is done by using CSMA/CA and using a CTS lasting much longer than the RTS. This ensures that hidden senders detect carrier while waiting for their CTS's. The tail of the CTS now becomes a busy tone from the receivers to all the senders, and all the senders back off enough to allow the data to go in the clear. Consequently, FAMA-NCS performs far better than CSMA and ALOHA. One method to improve the throughput of FAMA-NCS is to introduce multiple channels. In this scheme, all stations have a common control channel and different data channels. The RTS contention happens in the control channel. If the RTS reaches the receiver contention-free, the receiver switches to the data channel of the sender and sends back a CTS. The sender sends the data on the data channel after receiving the CTS. For the CTS and data to go in clear, it is essential for nodes two hops away to have different data channels.

2.2 Spread-Spectrum Techniques

In recent years, the use of spread-spectrum techniques for wireless networks has been an active area of research. CDMA is a direct application of spread-spectrum techniques. CDMA is different from the standard channel allocation strategies. Some of these standard strategies divide the channel into frequency bands and then assign them statically (FDMA)

or dynamically (wavelength division multiplexing). Others allocate the channel in bursts in time statically (FDMA) or randomly (ALOHA). CDMA allows each station to transmit simultaneously on a wide frequency spectrum. Using a wide bandwidth reduces multipath fading because of the frequency diversity achieved. It also increases tolerance to jamming and narrow band interference.

CDMA comes in two forms. In the frequency hopping (FH) systems the total bandwidth is partitioned into a set of channels of equal bandwidth [RKM95]. In this scheme, each stations is assigned a hopping pattern corresponding to its code. Both the transmitter and the receiver dwell on the same channel and hop together to another channel afterwards while talking to each other. Commercial applications of FH use *slow frequency hopping* in which the transmitter transmits more than one packet on a single channel. This requires less synchronization among stations than *fast frequency hopping.*

The second form of CDMA is direct sequence (DS). The spread spectrum signals are generated by linear modulation with wideband sequences (codes) assigned to each station. The spreading codes are the most important aspect of CDMA technology. The codes are Walsh codes based on a 64×64 Walsh matrix. Multiple transmissions can be separated using coding theory as explained below [Tan96].

In DS-CDMA each bit time is subdivided into m short intervals called chips. Each station is assigned a unique m-bit code or chip sequence. To transmit a 1 bit, a station sends its chip sequence or code. To transmit a 0 bit, the stations send the one's complement of it's code.

All chip sequences are pairwise orthogonal, which means that the normalized inner product of any two code's S and T is 0. This can be written as

$$S \bullet T \equiv \frac{1}{m} \sum_{i=1}^{m} S_i T_i = 0 \tag{2.1}$$

The normalized inner product of a code with itself is 1:

$$S \bullet S = \frac{1}{m} \sum_{i=1}^{m} S_i S_i = 1 \tag{2.2}$$

To get the bit stream of a sender, the receiver must know the sender's code in advance. The receiver does the recovery by doing the normalized inner product of the received signal and the code of the sender.

In an ideal CDMA network each station would have its own unique transmitting code. However, the number of orthogonal codes is limited. This gives rise to interference, because nodes in the neighborhood of one another may have the same codes.

Codes can be assigned in two ways in a CDMA network, the transmitter-oriented code assignment and the receiver-oriented code assignment. In a transmitter-oriented code assignment, the senders use their own code to transmit and the receivers tune into the sender's codes to receive. In receiver-oriented code assignment, the senders use the receiver's codes to transmit; the receivers merely listen on their code. *Secondary* interference can occur in both cases. Two stations unaware of each other's existence can transmit to the same receiver at the same time; which gives rise to the transmitter-oriented code assignment problem shown in Fig. 2.1. The codes of the station are shown in the brackets. Here, A and C need to have different codes and are two hops away from each other. The second case of secondary interference occurs when a station is transmitting to its neighbor and a third station's transmission to some station other than the stations involved in the first transmission causes an interference with the first transmission. This leads to the receiver-oriented code assignment problem illustrated in Fig. 2.2. Here, C is transmitting to D but causes interference at B, because B and D which are two hops away share the same code. As can be seen from the earlier two examples, both transmitter-oriented and receiver-oriented code assignment have the same requirement, i.e., no set of stations two hops away from one another can have the same code.

2.3 Approaches to channel and code assignment

Several approaches have been proposed in the past for channel/code assignments. Most of the papers dealing with channel assignment try to get optimal channel use in time by creating optimal TDMA cycles. This means that all nodes in the network share the same

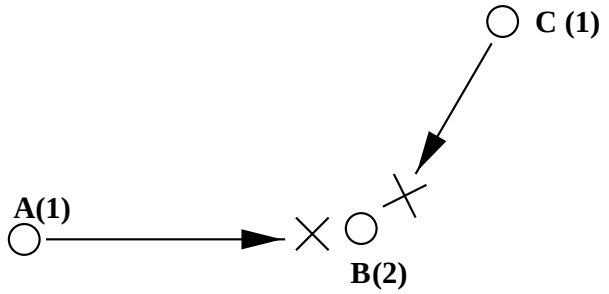


Figure 2.1: Interference when two transmitters two-hops away have the same code

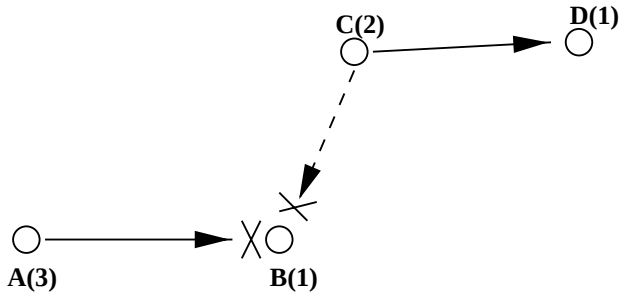


Figure 2.2: Interference when two receivers two-hops away have the same code

channel and therefore need to be scheduled such that there are no collisions among stations. The topology of the network is of major consequence, because stations that cannot hear one another and cannot potentially destroy each other's transmissions should be allowed to transmit at the same time. This is slightly different from our problem of assigning different frequencies to different stations. However, all solutions for getting a TDMA cycle can be mapped onto the problem of assigning the least number of frequencies correctly in a network, because the requirements for a solution are similar.

Chlamtac and Kutten [CK85] presented one of the first papers to address the problem of creating a TDMA cycle in a multi-hop network. They prove that obtaining an optimal assignment, such that the schedule is of smallest duration, is an NP-hard problem, and presented a greedy heuristic that uses the spanning tree created by a network-level protocol. This work was further extended in a later paper, where they presented the distributed version of their protocol [CK87]; the distributed version uses a traveling token, which incurs more overhead traffic on the network. Since they set up a static schedule, they

are able to bound the delay a broadcast packet will suffer before it reaches all stations in a network. Hajek and Sasaki presented two algorithms for obtaining fixed schedules in a spread spectrum network [HS88]. Their first algorithm finds a schedule of minimum length that allows each pair of neighboring stations to converse for a prescribed length of time. The second algorithm builds a link schedule that meets a prescribed end-to-end demand in a schedule of smallest length. A different approach by Cidon and Sidi consists of using control segments during which the schedule for the transmission segments is decided [CS88]; in their model secondary conflicts are not permitted unlike the model by Hajek and Sasaki, where only primary conflicts are taken into consideration. Ephremides and Truong [ET90] presented a distributed solution to the scheduling problem requiring stations to exchange schedules and require each station to know the schedule of stations one-hop and two-hops away. Ramanathan and Lloyd mapped the TDMA cycle problem to the graph coloring problem [RL93]. They also provided polynomial algorithms that improved the TDMA cycle length. In their analysis, the length of the schedule depends on the *thickness* of the network and not on the maximum degree of the network, as was the case in earlier papers. Thickness is a measure of planarity. It is defined as the minimum number of planar graphs a given graph can be split into. However, their algorithms are not distributed and work only for an one-time schedule. All the methods described above create schedules based on topology. Chlamtac and Farago [CF94] presented an algorithm which is independent of the detailed topology. Instead, the schedule depends on global parameters like, the number of nodes and maximum degree a node can have.

Transmitter-oriented code assignment is introduced by Makansi [Mak87]. Here, quasi-orthogonal codes are assigned to transmitters in a packet-radio network in such a way that hidden terminal interference is eliminated. A bound on the minimum number of codes for correct code assignments is derived. Code assignment was proven to be NP-hard in the paper by Bertossi and Bonuccelli [BB95]. Hu presented a technique using pairwise code assignment in which both the transmitter and receiver are assigned the same code.

This thesis presents a distributed channel/code assignment algorithm. This algorithm

assigns a channel/code to each node in a way that no interference occurs after the algorithm converges, provided that the number of channels/codes available for assignment is at least $d(d-1)+2$, where d is the maximum number of one-hop neighbors any node can have. This is because the algorithm assigns a data channel/code to a node that is different than the channels/codes assigned to nodes two hops away from the node. The algorithm is designed to be part of the MAC and routing protocols of a multihop packet-radio network. It is based on the asynchronous exchange of control messages that are part of the regular MAC and routing messages, and the information generated by any one node propagates up to two hops away from the node.

3. Code Assignment and Dissemination Algorithm

In this chapter, we mean both channel and code every time we use the word code. The same algorithm applies for both cases.

3.1 Network Model and Assumptions

To describe the code assignment and dissemination algorithm (CADA), we use the term "code" to mean channel, code or time slot in the context of assignment. We model a multihop packet-radio network as an undirected graph $G = (V, E)$ where V is the set of nodes and E is the set of edges. Each node consists of a transceiver and a router. A link between two nodes i and j in G means that i can hear j's transmission and j can hear i's transmission. Each node uses an omnidirectional antenna for transmission and the network works in half-duplex mode, which means that a node cannot transmit and receive at the same time. A routing protocol is assumed to create and update the routing table used at each node. This routing protocol provides information about who the node's active neighbors are; this involves adding to the neighbor list new neighbors when they come up and deleting neighbors that are no longer active. The routing protocol is assumed to have some form of neighbor discovery mechanism such as a HELLO exchange [MGLA96]. Nodes process messages they receive in FIFO order and links transmit packets in the FIFO order.

We also assume a MAC protocol that informs CADA of the code it hears the neighbors using. The floor acquisition multiple access (FAMA) family of protocols [FG95] is the example we use to illustrate the use of CADA with a MAC protocol. In FAMA, a sender transmits a request-to-send (RTS) to the receiver, which in turn transmits a clear-to-send (CTS) if it obtains the RTS free of errors. The RTS lasts longer than the propagation delay and the CTS must be larger than the aggregate time incurred in an RTS, a maximum round trip propagation delay and a transmit-to-receive turnaround time, so that collisions due to hidden terminals are eliminated in the absence of erasures due to drastic node mobility. When the network is first brought up, all transmissions take place over a common signaling

code using FAMA. As the stations decide on their codes, the stations use different codes for data transmissions.

A link is assumed to exist between two nodes only if there is good radio connectivity and the update messages of the routing protocol can be sent reliably. Node failures are modelled as all links incident on the node failing at the same time. A moving node is considered attached to all nodes with which the node can exchange messages with a certain probability of success; a moving node becomes detached from nodes with which it cannot exchange messages with this probability of success. CADA runs in conjunction with the routing protocol, which in turn runs on top of the MAC protocol.

3.2 Principles of Operation

The messages of CADA are called the *Code Assignment Messages (CAM)*. These messages are sent as part of the messages exchanged in the network's routing protocol. These messages could be lost due to changes in radio connectivity. Reliable transmissions of *CAM*s is done with the help of retransmissions. After receiving a *CAM*, the receiver is required to acknowledge it to the node that sent it, by sending an explicit *ACK* message indicating that the *CAM* has been received and processed.

3.3 Information at Each Node

The following structures are needed for code assignment at each node.

- **Priority List**: Each node has a unique priority number assigned to it. This priority number allows two-hop neighbors with the same code to decide which one of them should keep the code and which must look for new codes. We decided to use the address of a node as its priority number, because MAC addresses are unique and are simple to use.

  The priority list contains the priority numbers of the node, its one-hop neighbors and its two-hop neighbors sorted in increasing order. Each entry in the list has a flag set

to 1 when the neighbor is one hop away or two if the neighbor is two hops away. The codes assigned to the node itself and its neighbors are also listed.

- **Code Assignment Message Retransmission List** (*CAMRL*): This list has one or more retransmission entries, where an entry is of the following nature.
  - The sequence number of the *CAM*.
  - A retransmission counter that is decremented every time the node sends a *CAM*.
  - An *ACK*-required flag that is actually a field of the size of the neighbor list. The bit corresponding to a neighbor is set if the *CAM* is yet to be acknowledged by the neighbor.

  The *CAMRL* permits a node to know which *CAM* is not acknowledged by some of the neighbors and needs to be sent again. The node retransmits a *CAM* when its retransmission entry in the *CAMRL* reaches zero. The retransmission counter of a new entry in the *CAMRL* is set equal to a small number (e.g., 3 or 4).
- **Unassigned Code List** (*UCL*): This list contains all the available codes, i.e. they are not being used by any of the node's two-hop neighbors. This is implemented as a linear array which can be indexed in constant time.

The pseudocode in Fig. 3.1 shows the variables and data structures used in CADA. The procedure *Code_Init* describes the intialization phase of the algorithm.

3.4 Information Exchanged among Nodes

A *CAM* propagates only from a node to its neighbors and no further. Each *CAM* contains

- The address and code of the node which is sending the *CAM*.
- The addresses and codes of the node's one-hop neighbors.
- *ACK*'s to earlier *CAM*s. An *ACK* entry specifies the source and the sequence number of the *CAM* being acknowledged.
- A response list of zero or more nodes that need to send an *ACK* for this *CAM*.

Variables and Data Structures used
in a node for Code Assignment

Priority Table : PT with entries for
1-hop and 2-hop neighbors.
Each entry has the following fields:
adr: Neighbor Address.
$mask$: Neighbor Mask.
$code$: Neighbor Code.
hop: Number of hops.
set to 0 for node
set to 1 for 1-hop neighbor
set to 2 for 2-hop neighbor
$dual$: Clique Flag
set to 0 by default
Code List : CL where each entry of the list
corresponds to a single code.
Each entry is marked as *assigned* or *unassigned*
MAX_CODE : The maximum number of codes/channels the
MAC layer supports

Procedure Code_Init$(adr, mask)$
This module initializes the data structures for
the Code Assignment Algorithm running at node i.
The two variables adr and mask define the
FAMA interface used by the node i.
begin
for each *code* in CL
$CL[code] \leftarrow unassigned$
end for each
pick random code $rcode$
$CL[rcode] \leftarrow assigned$
$hops \leftarrow 0$
$dual \leftarrow 0$
add i to Priority Table$(adr, rcode, hops, dual)$
send Indication to MAC layer about new code
end

Figure 3.1: PseudoCode – Algorithm Initialization

If a single CAM is not large enough to hold all the codes and addresses, the information can be split up into more than one CAM.

A CAM is sent in the following situations.

1. When a node comes up, it broadcasts a CAM to all its neighbors, indicating that its priority list consists of its own address and its own code.
2. When a node i detects a change of code by any of its one-hop neighbors, i makes the required changes in its priority list and sends a CAM to all its one-hop neighbors, including the one that changed the code.
3. When a node i finds that a one-hop neighbor j is no longer active, it drops j from the priority list. This information is then conveyed to all its one-hop neighbors by a CAM reflecting the changes.

3.5 Updating the Priority List

A node updates its priority list either after detecting a change of code in one of its one-hop neighbors or after receiving a CAM containing information about a change of code by its two-hop neighbors.

When a node notices a change of code in any of its one-hop neighbors, it makes the change in its own priority list and sends a *CAM* with its own address and code, and the address of all its one-hop neighbors. This set of actions is handled by the procedure *Recv_Nbr_Indication* shown in Fig. 3.2.

If a two-hop neighbor changes its code, then any of the following three situations may arise:

1. If the new code of the two-hop neighbor is not the same as i's code, then the new code in entered into the priority list in the entry corresponding to the two-hop neighbor.
2. If the new code of the two-hop neighbor is the same as i's code and the address of the two-hop neighbor is lesser than i's code, then i picks up a new code. The new codes of the two-hop neighbor and i are entered into the priority list.
3. If the new code of the two-hop neighbor is the same as i's code and the address of the two-hop neighbor is greater than i's code, then i retains its code and there is a temporary conflict, until the two-hop neighbor changes its code. The two-hop neighbor learns i's code because of the *CAM* sent by the one-hop neighbor of i. This conflict does not stop the data transfer in the network.

In the first two cases the *UCL* has to be updated. All the codes used by the two-hop neighbors are marked as unavailable in the *UCL*. Procedure *Recv_CAM* shown in Fig. 3.3 handles all the above cases.

3.6 Sending New and Retransmitted CAM's

Whenever a node i sends a new *CAM*, it must perform the following steps:

1. Decrement the retransmission counter of all the new entries in the *CAMRL*.
2. Delete the entries corresponding to entries in the new *CAM*.
3. Add an entry in the *CAMRL* for the new *CAM*.

If a certain *CAM* in the *CAMRL* has all its entries covered be a new *CAM* transmission, the old *CAM* is deleted from the *CAMRL*.

```
Procedure Recv_Nbr_Indication(nbr,code)
The MAC layer sends an indication up every time it hears
a neighboring node advertise its channel.
This indication consists of the neighbor's address nbr
and its code code.
begin
   if the address of nbr exists in PT then
      if ( PT[nbr].code is not the same as code ) then
         if( PT[nbr].hops is not equal to 1) then
            if ( PT[nbr].dual equals 0 ) then
               PT[nbr].dual ← 1
            end if
         end if
         PT[nbr].code ← code
         Send_CAM
      end if
      else
         if ( PT[nbr].hops is not equal to 1) then
            if ( PT[nbr].dual equals 0 ) then
               PT[nbr].dual ← 1
               Send_CAM
            end if   end if
      end else
   end if
   else
      add nbr to Priority Table
      PT[nbr].code ← code
      PT[nbr].hops ← 1
      PT[nbr].dual ← 0
      Send_CAM
   end else
end
```

Figure 3.2: PseudoCode – Receiving Neighbor information from lower layer

When the retransmission counter for a *CAM* retransmission entry expires, node i sends a new *CAM* with the same data as the old *CAM*. However, it has a new sequence number and a new response list. This new response list specifies the neighbors that did not acknowledge the *CAM* earlier. The old entry in the *CAMRL* is deleted and a new entry created for the new retransmission.

Using the above retransmission strategy, a node can keep sending *CAM*s, until all its one-hop neighbors acknowledge it. However, if a node gets no response from a neighbor after a timeout, it considers the neighbor dead and resets bits corresponding to this neighbor in the response lists of all the entries in the *CAMRL*.

```
Procedure Recv_CAM()
This procedure updates the Priority Table and the Code List
using the information in the Code Assignment Message.
begin
    for each entry c in the CAM
        if an entry for c.adr exists in PT then
            get the corresponding entry e from the PT
            if c.code equals the node i's own code then
                if e.adr is greater than the node i's address then
                    set flag ChangeCode
                end if
            end if
            if c.code not equal to e.code then
                CL[e.code] ← unassigned
            end if
            CL[c.code] ← assigned
            PT[c.adr].code ← c.code
            if ((c.hops + 1) ≠ e.hops) then
                PT[e.adr].hops ← (c.hops + 1)
                PT[e.adr].dual ← 1
            end if
        end if
        else
            add entry c to Priority List
            PT[c.adr].hops ← 2
            PT[c.adr].code ← c.code
            PT[c.adr].dual ← 0
            if c.code is same as i's code then
                if c.adr is greater than the node i's address then
                    set flag ChangeCode
                end if
            end if
            CL[c.code] ← assigned
        end else
    end for each
    if flag ChangeCode is set
        pick new random code ncode for i
        if new random code ncode is available then
            CL[ncode] ← assigned
            PT[i].code ← ncode
            send Indication to the MAC layer
        end if
    end if
end
```

Figure 3.3: Pseudocode – Receiving Code Assignment Message

```
Procedure Send_CAM()
This procedure broadcasts a Code Assignment Message
to all neighbors.
begin
   set the destination of CAM to the broadcast address
   for each entry e in the PT
      if ((e.hops equals 1) or (e.dual equals 1))then
         create entry c to add to CAM
         c.hops ← 1
         c.code ← e.code
         c.adr ← e.adr
         add entry c to the CAM
      end if
   end for each
   send packet
end
```

Figure 3.4: PseudoCode – Sending Code Assignment Message

3.7 Processing an ACK

The *ACK* entry bears the sequence number of the *CAM* it acknowledges. As soon as an *ACK* is received by a node, the node searches its *CAMRL* to find the entry with the required sequence number. If a match is found the node resets the *ACK* required flag for the neighbor which sent the *ACK*.

A node may receive an *ACK* for an entry which has been deleted due to more recent *CAM*s to the same neighbor or for an entry which has undergone a change in sequence number due to more recent retransmissions. In that case, the node simply ignores the *ACK*.

3.8 Special case of a three-way clique

The algorithm needs to be modified slightly to accommodate the presence of three-way cliques. An example is shown in Fig. 3.5. Here, A, B and C are both one-hop and two-hop neighbors to each other. Since they can all cause two-hop interference, they need to be treated as two-hop neighbors, i.e., all three of them need to have separate codes. However, A needs to report B and C to its neighbors D and E as one-hop neighbors as they can cause two-hop interference for D and E. We surmount this problem, by setting a special

flag if any nodes are part of a three-way clique. If the flag is set for a certain node, this node is treated as a two-hop neighbor while assigning codes but as a one-hop neighbor while forwarding information.

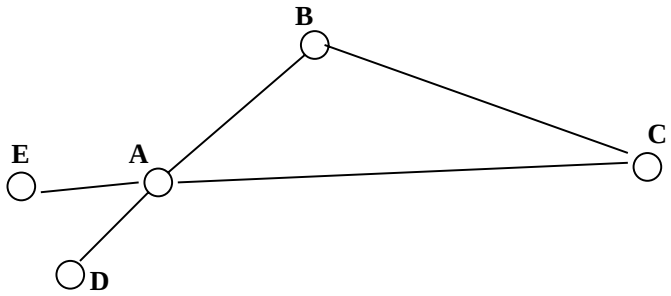


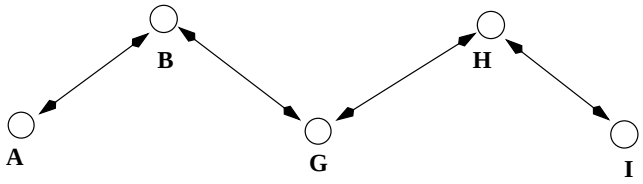
Figure 3.5: Example of a three-way clique

3.9 Example of algorithm

The working of the algorithm is shown in an example five-node network with ten available codes, as shown in Fig. 3.6. When node G comes up, it picks up a random code for itself and sends an initial CAM consisting of only its address and code. After the MAC protocol hears the codes used by the one-hop neighbors, it informs the algorithm. The CAM now has two more entries which consist of the address and code of one-hop neighbors B and H as shown in Fig. 3.7. Fig. 3.8 shows the priority table at node G after it hears CAMs from its one-hop neighbors. Node G realizes that it has the same code as two-hop neighbor A. Since node G has a higher address it picks up a new code as shown in Fig. 3.9. As can be seen from all the figures, the unassigned code list contains all the codes not being used by the node or its two-hop neighbors.

3.10 Embedding Code Assignment Algorithm in MAC and Routing Protocols

The proposed code assignment algorithm can be used as an integral part of the MAC and routing protocol of a packet-radio network.



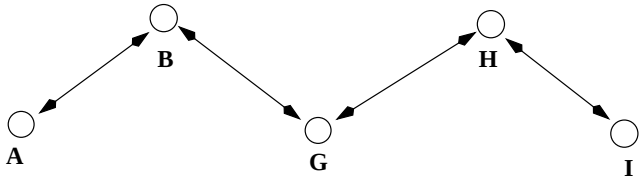
G's Code Assignment Message

G	6

G's Unassigned Code List

1	2	3	4	5	7	8	9	10

Figure 3.6: Initial code assignment message



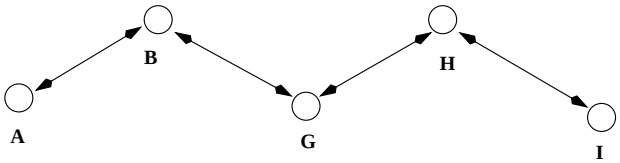
G's Code Assignment Message

B	2
G	6
H	9

G's Unassigned Code List

1	2	3	4	5	7	8	9	10

Figure 3.7: Message sent after figuring out one-hop neighbor's code



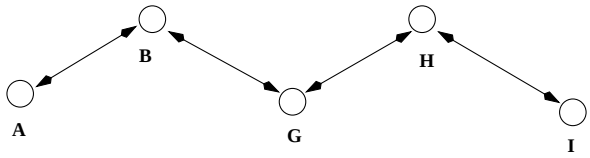
G's Priority Table

A	6	2
B	2	1
G	6	0
H	9	1
I	1	2

G's Unassigned Code List

2	3	4	5	7	8	9	10

Figure 3.8: Priority table after receiving information about one-hop and two-hop neighbors



G's Priority Table

A	6	2
B	2	1
G	3	0
H	9	1
I	1	2

G's Unassigned Code List

2	4	5	7	8	9	10

Figure 3.9: Priority Table after changing code

In our model, all stations have a common signaling code on which they implement an RTS-CTS exchange similar to the 802.11 protocol [IEE91]. The data transfer can be done on the receiver's code for $ROCA$ or on the transmitter's code for $TOCA$. The advantage of some form of code assignment is that a node does not require all other nodes in its one-hop neighborhood to be silent while it receives a transmission. This increases usage of bandwidth.

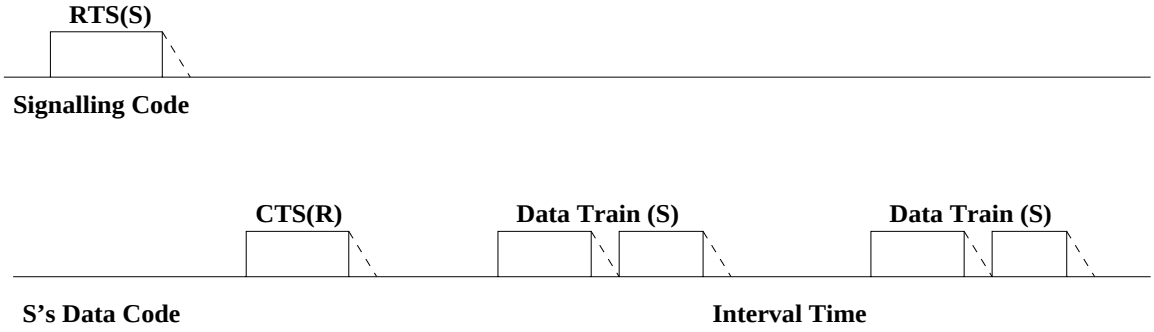


Figure 3.10: Handshake and Data Transfer using Transmitter's Code

We investigate the case for the transmitter-oriented approach in which we use the sender's code for data transfer (Fig.3.10). Consider two nodes S and R that are immediate neighbors of each other. The senders of the packets are specified in the parentheses. Node S needs to transmit data to R and sends an RTS to R specifying its transmission code and the number of data packets it plans to send. The RTS is sent on the signaling code. After sending its RTS, node S shifts to its own data code and listens. R send back a CTS on S's data code. The CTS contains the maximum number of data packets it can allow. After successfully receiving the CTS, node S transmits its data on its data code.

After the specified number of data packets have been transmitted and received, S and R switch back to the signaling code. If the RTS-CTS exchange also specifies the interval time between two packet bursts, then S can switch to its data code automatically after the initial handshake, without an intervening RTS-CTS handshake. The last packet of a data burst can contain control data specifying if there are more data bursts following. If node R gets any requests that would require it to receive data from any node other than R during the

time it has reserved for R, it refuses the requests. Also, any RTS arriving(on the signaling code) at R while it is receiving data from S (on S's data code) would not destroy the data. This takes care of the multihop problem.

Now, with the availability of assured time for data transmission, one can visualize guaranteeing a certain rate of data transmission between two neighbors. This is useful for real-time data traffic, which requires constant delay. However, one has to keep in mind that if the topology changes constantly, such a rate may be sustained, if the code assignment has to be modified as a result.

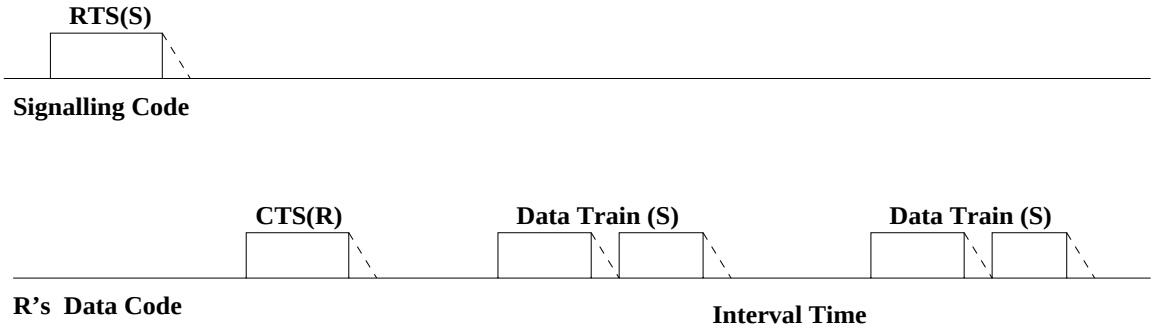


Figure 3.11: Handshake and Data Transfer using Receiver's Code

A similar treatment can be done for the receiver oriented approach shown in Fig. 3.11. Here, the RTS is sent on the signalling code but the CTS and data are sent on the reciver's data code.

A *CAM* must contain information about a node's one-hop neighbors and their codes. If the distance metric in a routing algorithm is hop-count, then reading a routing update message from a node is enough for the code assignment algorithm to deduce a node's one-hop neighbors. However, the codes assigned to the neighbors are not present in the routing updates. With the addition of this field, the routing updates can be used as *CAM*s.

4. Correctness

This section shows that the code-assignment protocol is correct under the assumption that an underlying protocol assures the following conditions:

1. A node detects within a finite time the existence of a new neighbor or the loss of connectivity with a neighbor. A node also detects within a finite time a change in the code used by a neighbor.
2. All messages transmitted over a radio link are received correctly and in the proper sequence within a finite time.
3. All messages and code changes are processed one at a time within a finite time and in the order in which they are detected.

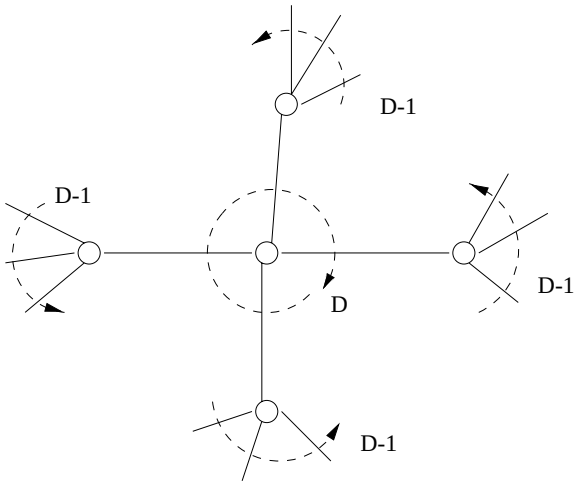


Figure 4.1: The connectivity of a node in a network a maximum degree d

Another assumption we make is that we have a minimum of $d(d-1)+2$ codes, where d is the maximum degree of the network. One code is required for signaling and the use of the rest of them can be explained using Fig. 4.1. A node can have a maximum of d neighbors. Each of these neighbors can in turn have d neighbors. Consider the central node i. It has d one-hop neighbors and $d(d-1)$ two-hop neighbors. The node should not have a code in common with any of its two-hop neighbors. Thus, it is clear that we require $d(d-1)+1$

codes. This is a sufficient condition to obtain valid code assignments [Hu93]. The network can still operate with fewer than $d(d-1)+2$, even though some two-hop neighbors have the same codes. However, for the proof, we assume that $d(d-1)+2$ codes exist.

Three additional assumptions are that there are a finite number of topology changes up to time t_0, and that no more changes occur after that time, and that nodes can correctly determine which *CAM*s are more recent than others, and that there are a finite number of nodes in the network.

Correctness for this algorithm means that, within a finite time after t_0, all nodes obtain information about the codes of their one-hop and two-hop neighbors. This allows them to calculate a code for themselves that is different from the codes of all their two-hop neighbors.

There are two different ways a node can change its code:

1. The node can change its code to any of the codes unused by its two-hop neighbors (this includes codes of one-hop neighbors).
2. The node can change to a code that is already being used by a two-hop neighbor with an address greater than itself.

A node is said to have consistent information in its priority table if it has the most recent information about the change of codes of its one-hop and two-hop neighbors.

Theorem 1: *A finite time after t_0, all nodes have consistent information in their priority tables and the codes they select based on this information are such that no two-hop neighbors have the same code.*

To prove the above statement we need to show that the following conditions are satisfied:

1. All nodes eventually stop updating their priority list and stop sending update messages to their neighbors.
2. All nodes must have consistent code information in their topology databases within a finite amount of time after t_0.
3. If the information in the node is consistent and most recent, a node only makes a valid change of code and there are no deadlocks.

4. If all nodes make only valid change of codes, then we have a correct code assignment, such that no pair of two-hop neighbors have the same code.

Lemma 1: *All nodes eventually stop updating their priority list and stop sending update messages to their neighbors.*

Proof: First, note that there are a finite number of nodes in the network, and that by assumption a finite number of code changes can occur up to time t_0, after which no more topology changes occur. Also by assumption, a change of code by a certain node is detected within a finite amount of time by the neighboring nodes. These neighboring nodes, in turn update their priority lists and send out at most one CAM to each of their neighbors. Therefore, for any change of code, there can be at most $d(d-1)$ messages sent.

A node x that does not allow the protocol to terminate must be generating an infinite amount of CAMs. This is only possible if one of its neighbors $x-1$ is changing its code an infinite number of times. For $x-1$ to change its code an infinite number of times, it has to get an infinite number of CAMs from a one-hop neighbor $x-2$. In turn, $x-2$ would send infinite number of CAMs only if a one-hop neighbor $x-3$ changes its code an infinite number of times. Because the network is finite and node ID's are unique, it follows from the code-assignment algorithm operation that there is always a node with the lowest ID that never changes its original code. Accordingly, it is impossible to continue with the same line of argument and the protocol can produce only a finite number of CAMs; therefore, the message transfer must stop within a finite time after t_0.

Q.E.D.

Lemma 2: *All nodes must have consistent code information in their topology databases within a finite amount of time after t_0.*

Proof: Consistent code information means that the node knows about all the recent code changes in its one-hop and two-hop neighbors. Consider some node i; a lower-level protocol guarantees that i detects a change of code in any of its one-hop neighbors within a finite time.

Consider the case of a node i whose two-hop neighbor k changes its code. This two-hop neighbor is the one-hop neighbor of at least one of i's one-hop neighbors (j). It is true that j detects a change in k's code in a finite time. This detection causes a change in j's priority list, which in turn causes j to send a CAM to all its one-hop neighbors including i and k. Because, we assume that node j processes CAMs in a finite time, we must have the latest code change information in a finite time after the last change, i.e., after t_0.

Q.E.D.

Lemma3: *If the information in the node is consistent and most recent, a node only makes a valid change of code and there are no deadlocks.*

Proof: When a node receives a CAM, it makes changes in its priority table and determines whether any of its two-hop neighbors use the same code as the one it uses. If no two-hop neighbor has the same code, the node does nothing. If a two-hop neighbor has the same code, then the node checks the address of the two-hop neighbor. If the two-hop neighbor has a larger address, then the node keeps its own code. If the two-hop neighbor with the same code has an address smaller than the node, then the node has to pick up a new code from the set of unassigned codes. If there are $d(d-1)+2$ codes, there always is some code available to pick. Therefore, there is always a valid change of code, that can take place in a finite time. Furthermore, there exist no deadlocks because there are no cases in which a node has to wait forever for the availability of a code.

Q.E.D.

Lemma 4: *If all nodes make only valid changes of codes, then we have a correct code assignment in which no two-hop neighbors have the same code.*

Proof: Consider a set of two-hop neighbors $a_1, a_2...., a_n$ sorted with respect to their addresses. Under the condition of valid change of codes, no node can have the same code as a two-hop neighbor with lesser address, i.e, a_2 cannot have the same code as a_1, a_3 cannot have the same code as a_1 and a_2. If the condition of valid change of codes is true for all nodes, no node a_i can have the same code as $a_{i-1}, a_{i-2}, ..., a_1$. If we continue this argument until reaching a_n, we see that a_n cannot have the same code as $a_{n-1}, a_{n-2}...a_1$. From this

we see that, for a set of two-hop neighbors with a bounded number of nodes, no two nodes have the same code. One of the assumptions we made is that we have a finite network, i.e. a bounded number of nodes. Therefore, we have a correct code assignment in which no two-hop neighbors have the same code.

Q.E.D.

5. Complexity of Protocol

Communication Complexity is the number of messages exchanged by CADA in the worst case. Messages are sent when a node detects a change of code by any of its one-hop neighbors.

There can be two results of a node changing its code.

1. The node's new code is not the same as any of its two-hop neighbors' codes. In this case, there are only $O(d^2)$ messages exchanged because each one-hop neighbor sends CAMs to all its one-hop neighbors.
2. The node's new code is the same as one of its two-hop neighbors' code. In this case, the two-hop neighbor might have to change code, which in turn could cause CAMs to be sent. In a pathological case, a node's change of code might cause all the nodes in the network to change their codes.

From the above, it follows that the number of messages is bounded by $O(|V|.d^2)$.

Complexity of Computation is the number of steps it taken when the algorithm is invoked at any node. The most important routines to analyze are *Recv_Nbr_Indication* and *Recv_CAM*. *Recv_Nbr_Indication* is called when the algorithm receives an indication from a lower layer about the code of a one-hop neighbor. If the code has changed or a new neighbor is discovered, the node sends a CAM. This operation of sending a CAM requires a linear scan of the priority list which is of the order of $O(d^2)$. *Recv_CAM* reads each entry in a CAM and performs certain actions. There are $d+1$ entries in the CAM, in the worst case. This includes new entries that did not exist in the priority list earlier. The entries in the CAM are presumed to be in the sorted order. A scan of the CAM is done and the new nodes are added to the priority table. If a new code is used this code is marked as assigned in the unassigned code table. Because the priority list and the unassigned code table can be implemented as hash tables accessible in constant time, *Recv_CAM* takes time proportional to $d+1$. While reading each of the entries, we can check if any of the two-hop neighbors

of the node have the same code as the node. If they, do a flag is set. After reading all the entries, if the flag is set, the node needs to find a new code. In the worst, this can involve a linear search of the unassigned code list which is of size c. Thus the computation complexity of the protocol is $O(d + c)$, and we know from earlier discussions that c needs to be at least $O(d^2)$; therefore, the complexity is $O(d^2)$

Storage Complexity is the amount of memory the major structures in the algorithm use. The priority list has $d(d-1)+d+1$ entries. The MRL can have a specified maximum number of entries which is presumed to be a constant. The UCL has to have a minimum of $d(d-1)+2$ entries. Therefore, the storage complexity is $O(d^2)$.

6. Simulation

In this chapter, we present the simulation results for CADA, which was described and proved correct in Chapters 3 and 4 respectively. The simulations presented here attempt to numerically quantify three main performance parameters: the gain in throughput achieved by using the code assignment algorithm, the scalability of the code assignment algorithm and the adaptability of the algorithm to mobility.

The rest of the chapter is organized as follows. Section 6.1 gives a brief introduction of the simulation tool used and discusses the metrics used to measure the performance of the algorithm. Section 6.2 presents the results for the throughput comparison. Section 6.3 describes the experiments used to measure the scalability of the algorithm. Section 6.4 presents the convergence time for varying degrees of mobility. Finally, Section 6.5 summarizes the chapter.

6.1 Implementation of CADA

The simulations of the CADA were done using the C++ Protocol Toolkit (*CPT*) [1] . The simulation environment is modelled to be as close to real-world as possible. This allows seamless transition of the simulation protocol software into an embedded hardware system.

CPT comes along with the CPF library which has pre-defined objects that can be used to reduce the amount of code that needs to be written. Some example objects are : *Node, Protocol, Protocol Shell, Device, Packet, Control Parameters, Buffer, Timer, Hash Table, Event, Queue, Priority Queue, Finite State Machine, Data Manager,* and *Log Manager.*

CADA is derived from the *Protocol* object. Protocol objects talk to each other through the Packet and Control interfaces. The Packet interfaces are used to send packets to lower or upper layers and to receive packets from them. The Control interfaces are used to send control messages up and down in the protocol stack. The structure of the protocol stack

[1] We thank Rooftop Communication for designing CPT.

for a wireless router with code assignment capability is shown in Fig. 6.1. This same stack is used in the simulations. WIRP is the CPT implementation of Wireless Internet Routing Protocol [MGLA96] and FamaIpIf is the CPT implementation of FAMA [FGLA97] with additional functionality added to interface with IP and handle IP packets.

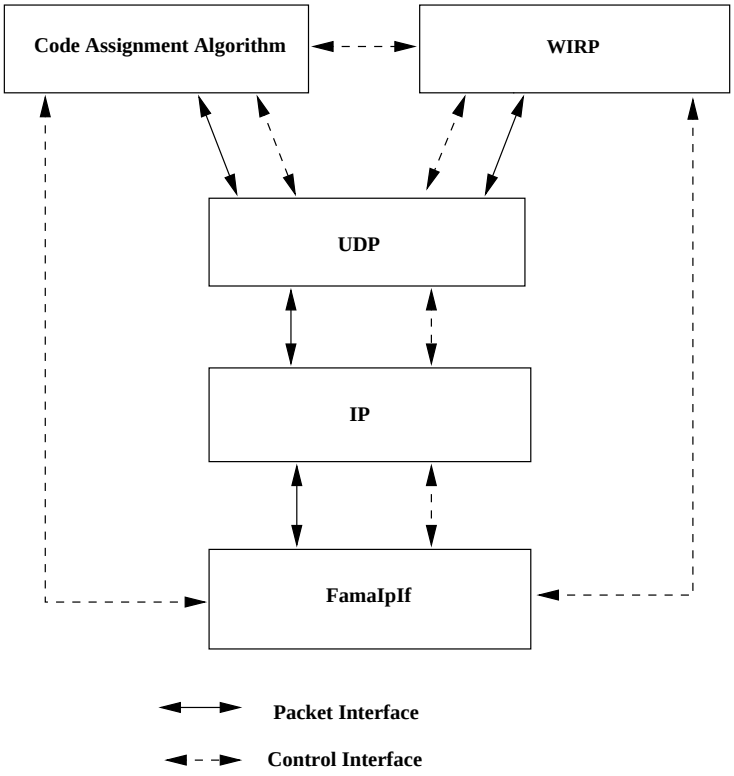


Figure 6.1: Stack used for implementation

The following are the metrics studied in the various experiments:

Throughput: Having correct data channel assignments should result in an increase in throughput. The first experiment is designed to measure the difference in throughput when there is correct code assignment, as compared to having wrong code assignment, i.e., nodes two hops away from each other having the same code. Throughput is defined as the number of packets received per unit time at the receivers. Throughput is measured at the MAC layers and thus includes both the control packets sent by the various upper layers and the data packets sent by the traffic generator. However, the rate of packets sent by the traffic

generator far exceeds the rate of control packets. Therefore, to see the performance of the MAC layer, we change the size and rate of packets sent by the traffic generator. Most experiments are done for two types of data traffic - Poisson and periodic.

Number of messages: In the complexity analysis, we stated that the number of code assignment messages increases linearly with the maximum connectivity of the network. This is verified in the experiments presented in Section 6.3 and Section 6.4. We measure the number of control messages sent and received until the algorithm converges. CADA converges when every node has a code that is different from the codes of all its two-hop neighbors.

Events: This metric involves measuring the total number of changes to the priority table. This includes counting the number of additions of nodes and changes in the codes and number of hops of the nodes already present in the priority table.

Time for convergence: The time it takes from the start of the simulation to the time nodes have correct code assignments and there are no more event-driven messages in the network.

Both the time and number of messages are measured in networks that have no data traffic. We measure the last two metrics in experiments testing scalability and mobility adaption.

6.2 Results-Throughput Difference

In this section we see results from experiments designed to test the increase in throughput resulting from using code assignment. The topology for the first two experiments is the simplest possible case of two-hop channel interference. The topology is shown in Fig. 6.2. This is a four node network running the multichannel version of FAMA. The dashed lines show the connectivity in the topology. Therefore, node 2 can only hear node1 and node3. Node4 acts as a hidden terminal to node 2. Each of the radio links has a capacity of 1 Mb/s and has zero bit error rate. This implies that control RTS packets are lost only due to collisions and data packets are lost only due to interference by a transmission from a

node two hops away. There is no packet loss caused by fading, random noise and other such channel problems. The simulation incorporates noncapture, i.e., the overlap of any fraction of two packets results in destructive interference and both packets must be retransmitted.

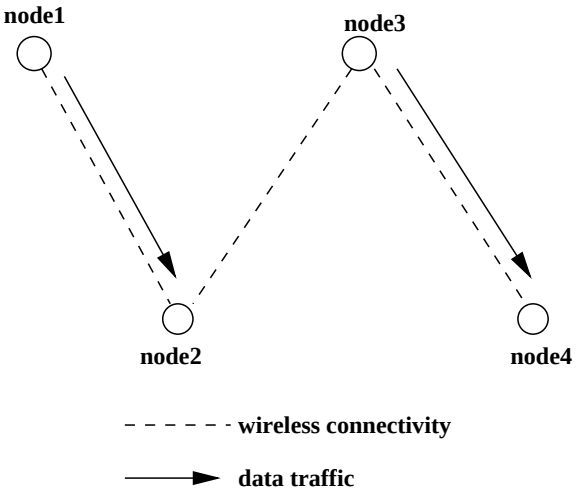


Figure 6.2: Topology used for throughput simulation

There are two streams of data traffic. One from node 1 to node 2 and the other from node 3 to node 4. We compare two situations. The first is when there is correct data channel assignment and the second is when there is wrong data channel assignment. For the wrong data channel assignment, all the nodes are assigned the same data channel. Two types of data traffic are considered, periodic and Poisson.

6.2.1 Experiment 1

For the first experiment, the receiver throughput at node 2 is considered, and we measure the number of Kbytes of data received per second by node 2. The interarrival time of the data is varied from 0.1 seconds to 0.01 seconds. The size of all data packets is 435 bytes. All the measurements are done after the network reaches steady state, i.e., there are no changes in the topology of the network and therefore there are no event-driven control messages being sent. From Fig. 6.3 and Fig. 6.4, we can see that correct data channel assignment results in increased throughput. In both kinds of traffic the throughputs plateau after a certain data

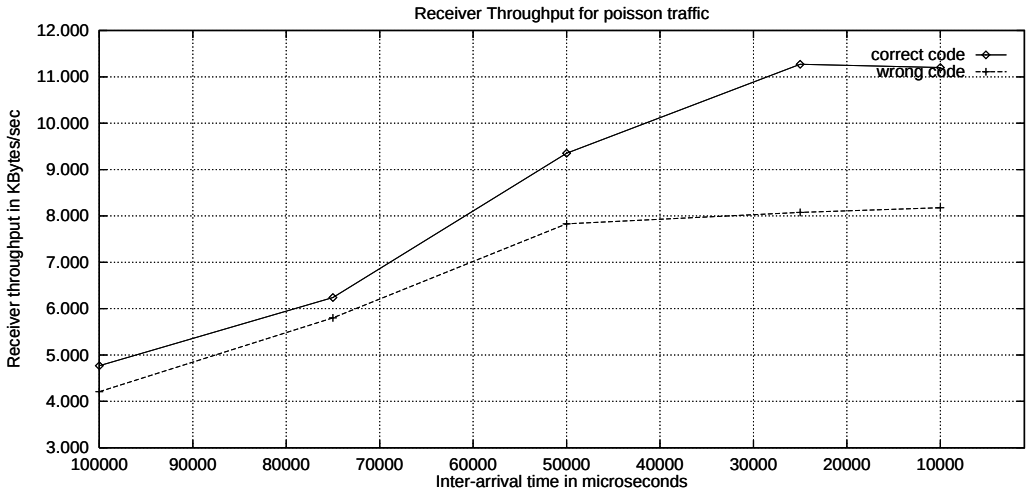


Figure 6.3: Receiver throughput at node 2 with varying rates of Poisson traffic, Packet Size = 435 bytes

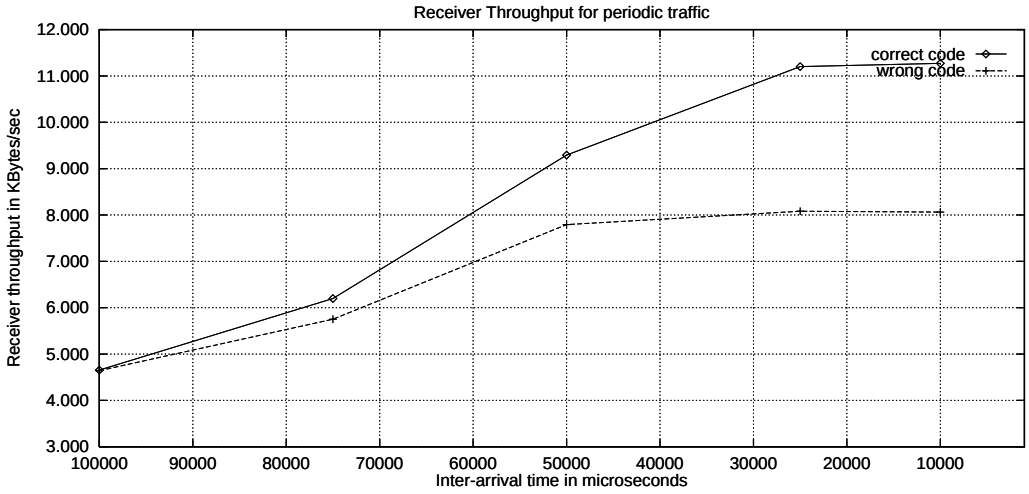


Figure 6.4: Receiver throughput at node 2 with varying rates of periodic traffic, Packet Size = 435 bytes

rate is reached. This is due to increased number of collisions. However, the throughput from correct channel assignment always saturates at a higher data rate, approximately 39% higher.

6.2.2 Experiment 2

This experiment uses the same set-up as Experiment 1, but varies the data packet size. The sizes tested are 500, 700, 900 and 1100 bytes, respectively. The interarrival time of the packets is kept constant at 0.05 seconds. The results of correct and wrong data channel assignment are plotted.

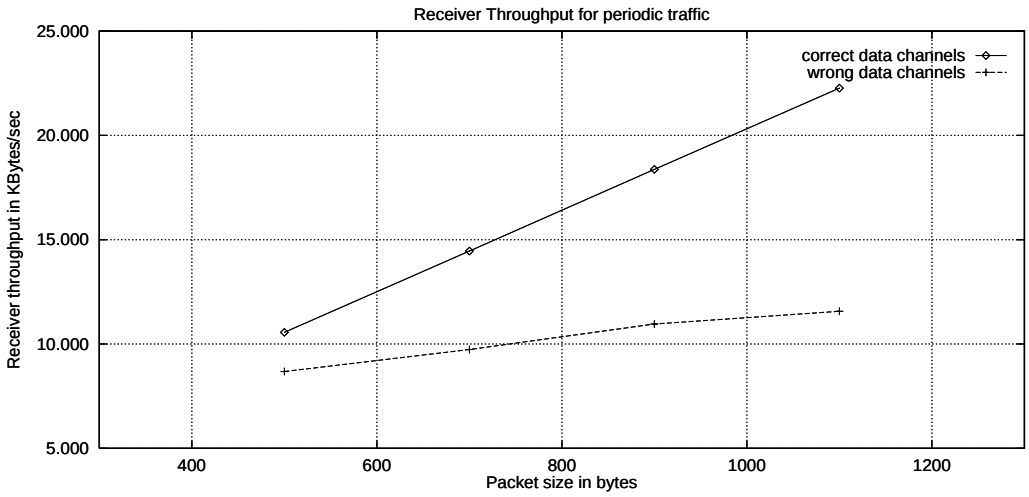


Figure 6.5: Receiver throughput at node 2 with varying packet sizes, Average Interarrival Time = 0.05 seconds

The results are shown in Fig. 6.5 and Fig. 6.6. It is apparent that using correct channel assignment allows the throughput to increase as the packet size increases. The wrong channel assignment does not see as great an increase in throughput because if there is two-hop interference while a larger data packet is being transmitted a greater amount of data is lost.

6.2.3 Experiment 3

This experiment uses a different network topology than the previous two experiments. The network topology is shown in Fig. 6.7. The experiment consists of varying the number of transmitting one-hop neighbors that a receiver (in this case $R0$) has. Accordingly, when

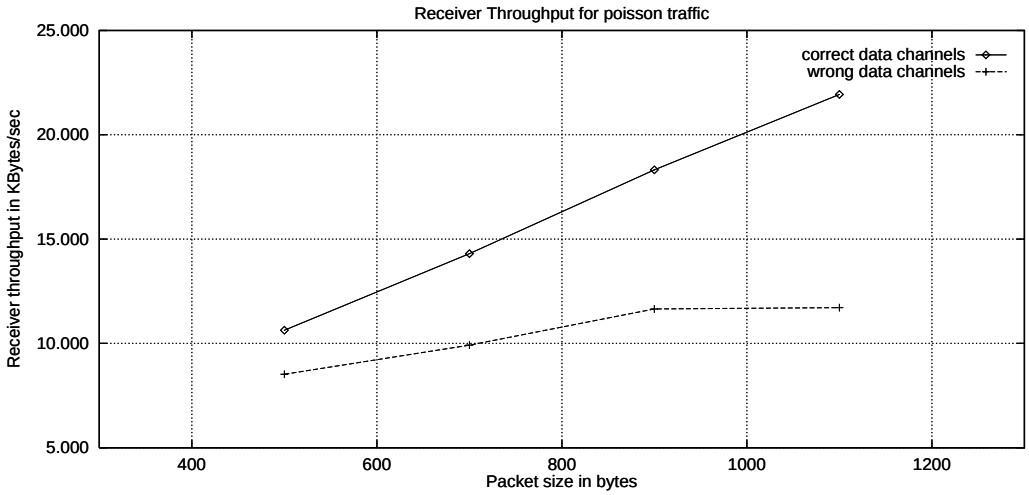


Figure 6.6: Receiver throughput at node 2 with varying packet sizes, Average Interarrival Time = 0.05 seconds

there is only a single one-hop neighbor, the network consists of nodes $S0, R0, S1$ and $R1$. For two one-hop neighbors the nodes $S2$ and $R2$ are added, and so on.

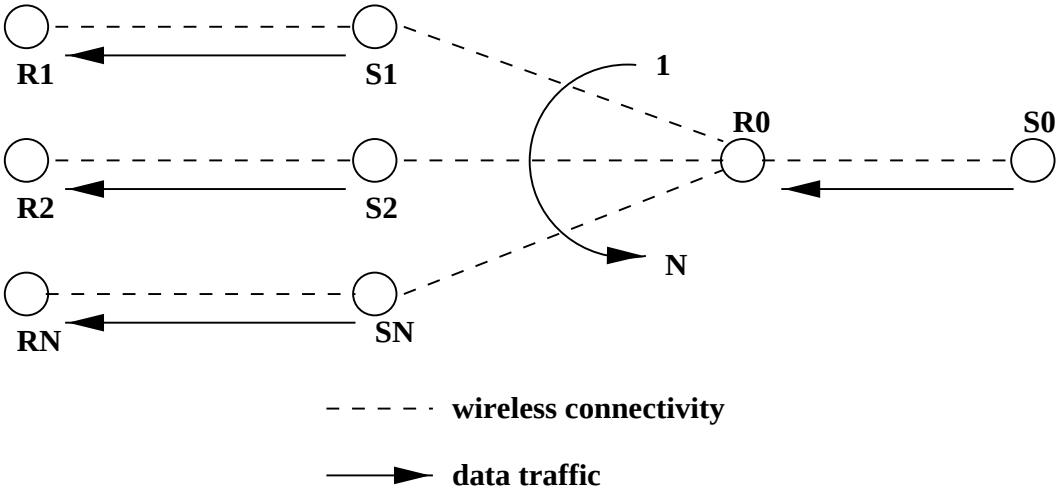


Figure 6.7: Topology for throughput simulation with varying number of neighbors

The packet size is 500 bytes and the data traffic is Poisson. Results have been plotted for two traffic rates, one with an interarrival time of 0.05 seconds and the other with an interarrival time of 0.1 seconds.

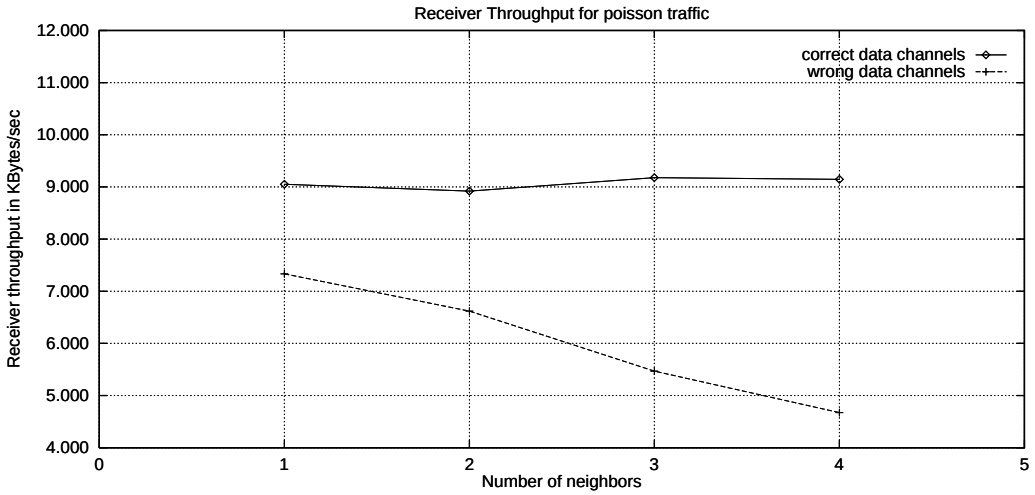


Figure 6.8: Receiver throughput at node 2 with varying number of 1-hop neighbors, Average Interarrival time = 0.1 seconds

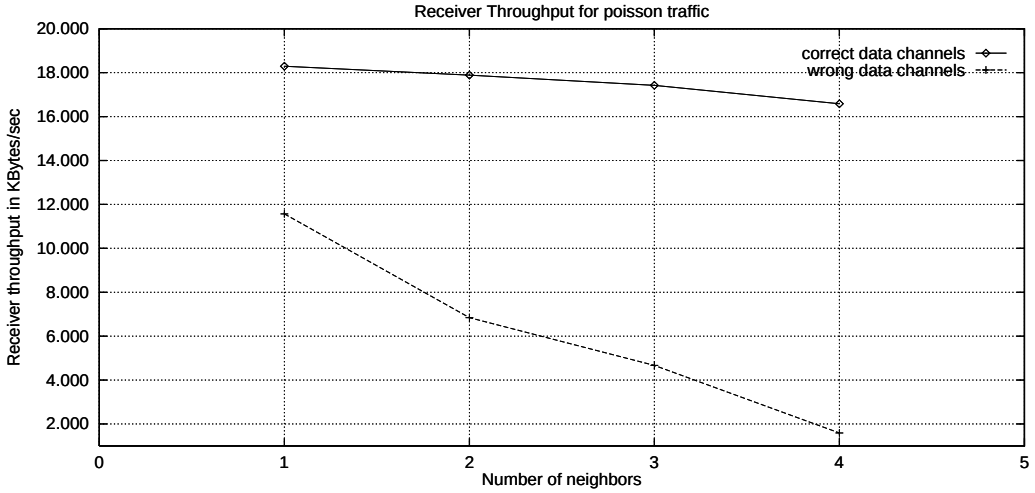


Figure 6.9: Receiver throughput at node 2 with varying number of 1-hop neighbors, Average Interarrival time = 0.05 seconds

Fig. 6.8 plots the case of the low data traffic rate. We can see that the throughput in the network with wrong code assignment falls drastically as the number of neighbors increases. This basically corresponds to the case of pure ALOHA. On the other hand, the network with correct code assignment is not affected by the number of neighbors. Besides correct code assignment, another reason for better performance, is that at low data traffic rates

there are not many RTSs colliding. Fig. 6.9 shows the case of the higher data traffic rate. The throughput for wrong code assignment plummets as was seen in the low traffic rate. However, the throughput for correct code assignment falls too, indicating increased RTS collisions.

6.3 Scalability

This section describes the results of the experiments used to measure the scalability of the algorithm. Scalability can be measured using three metrics: number of *CAM*s, events and the time it takes for the algorithm to converge. The first experiment runs the code assignment algorithm on a static network and compares values for nodes with different connectivity. The second experiment runs the algorithms on networks of differing maximum connectivity.

6.3.1 Experiment 1

The code assignment algorithm is run on the graph shown in Fig. 6.10. There are six nodes with one one-hop neighbors, six nodes with two one-hop neighbors, two nodes with three one-hop neighbors, four nodes with four one-hop neighbors and four nodes with five one-hop neighbors. The values are collected for each of the node and then the values for nodes with the same one-hop connectivity are averaged. The results are shown in Fig. 6.11 and Fig. 6.12.

We can see from the graph in Fig. 6.11 that the number of messages increases with increased connectivity. There is almost a linear relationship between messages and one-hop connectivity. The variation from a linear relationship results from the presence of 3-way cliques, i.e, nodes that are both one-hop and two-hops away. Fig. 6.12 shows that the number of events also increases with the increasing connectivity.

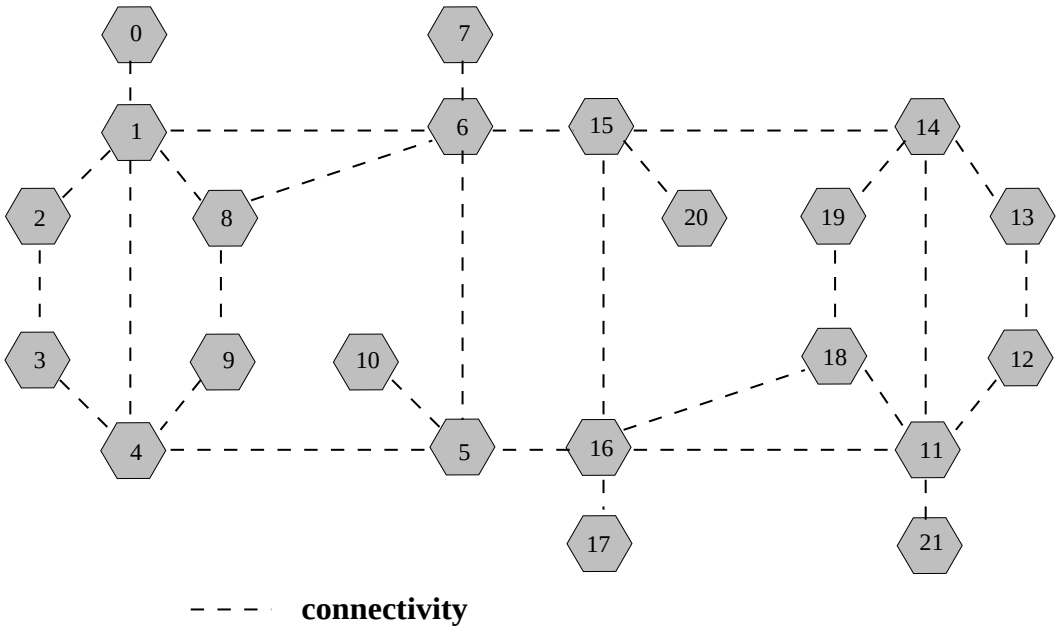


Figure 6.10: Network topology for the scalability experiment

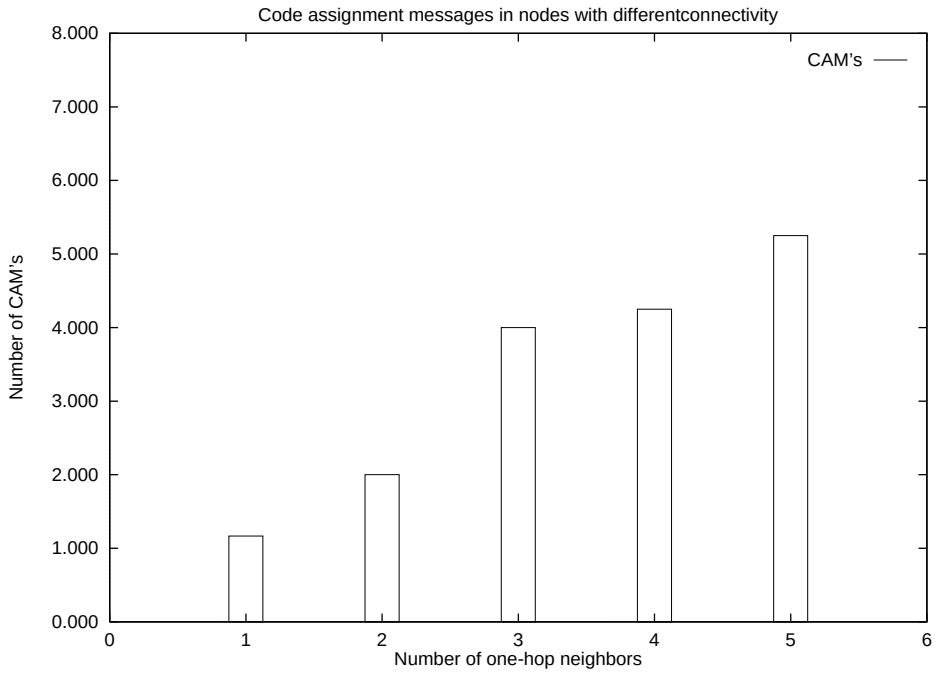


Figure 6.11: Change in number of event-driven CAMs as one-hop connectivity increases

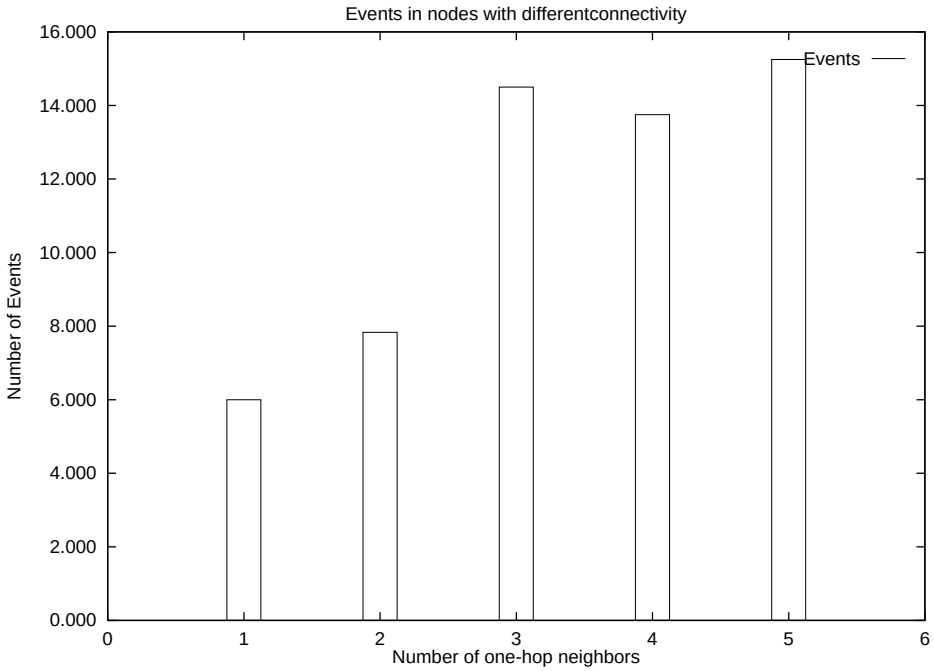


Figure 6.12: Change in number of events as one-hop connectivity increases

6.3.2 Experiment 2

This experiment attempts to compare the time it takes for the network to converge. We compare networks with varying maximum connectivity. The setup used in this experiment is the same as that used in Section 6.2.3. As the neighbors $S0$, $S1$, $S2$, $S3$ and $S4$ are added the maximum degree of network changes from two to five. There is no data traffic in this experiment. The transmission rates of all the nodes are constant at 1 Mb/s. Fig. 6.13 shows the results. The convergence time increases as the maximum degree of the network increases.

6.4 Mobility

The final experiment we did was to test the adaptiveness of the code assignment algorithm to mobility of the nodes. For this experiment, we used the graph with 30 nodes shown in Fig. 6.14. To introduce mobility, we used a utility in CPT that picks random nodes for movement. The path taken by the nodes is also random. The utility automatically adjusts

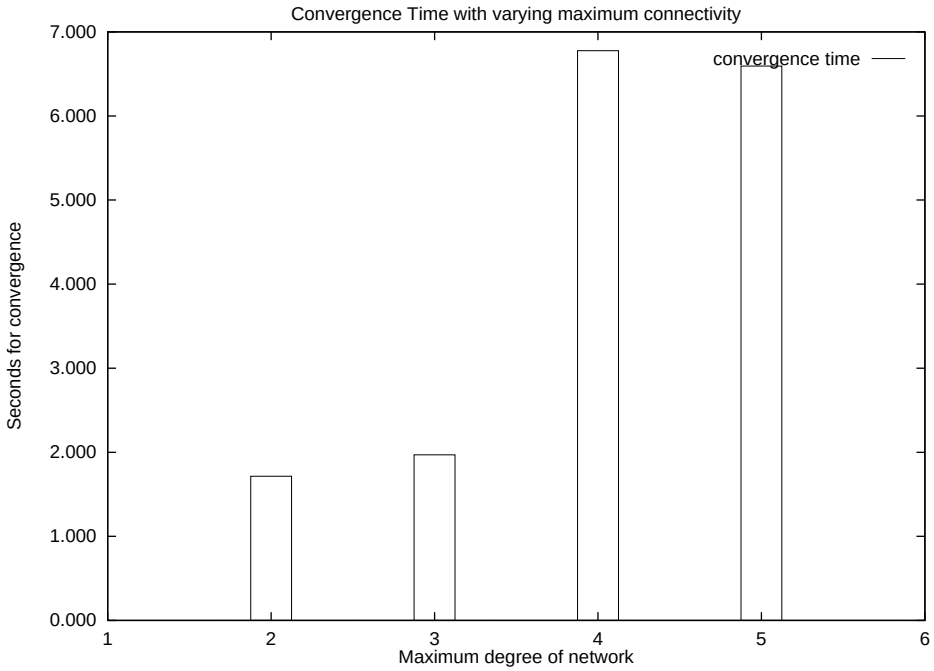


Figure 6.13: Convergence time for networks of varying maximum connectivity

the attenuation between the nodes to match the speeds at which the nodes move. We studied the mobility results for three cases; five nodes moving, 15 nodes moving and 25 nodes moving. For each of these cases we studied three speeds, 0.01 meters/sec, 0.1 meters/sec and 1.0 meters/sec. For each of the nine cases we generated 10 different mobility sets, i.e., a different set of random nodes would move each time. The transmission rate is 1 Mbps.

Number of Mobile Nodes	5	15	25
Speed in meters/sec	Average/σ	Average/σ	Average/σ
0.01	715.4/0.84	716/1.05	716.4/1.35
0.1	715.8/1.93	718.2/1.39	725.7/6.44
1.0	745/13.51	808.1/61.30	1125.2/550.61

Table 6.1: Number of event driven Code Assignment messages sent over a period of 1900 seconds

The results are shown in Table 6.1 and Table 6.2. We see that there is almost no change

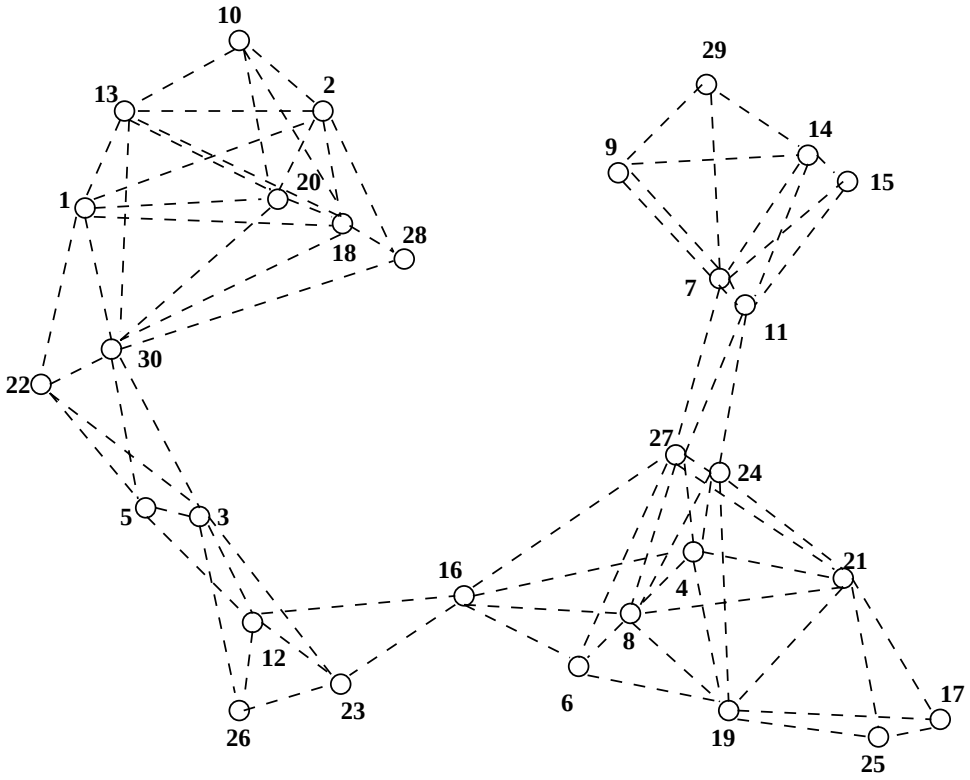


Figure 6.14: Initial 30 nodes topology for mobility testing

Number of Mobile Nodes	5	15	25
Speed in meters/sec	Average/σ	Average/σ	Average/σ
0.01	1933.6/1.35	1934.2/1.40	1935.2/2.20
0.1	1934.6/2.80	1940.2/5.181	1958.7/16.40
1.0	2039.7/58.52	2256.1/227.59	3222.1/1511.02

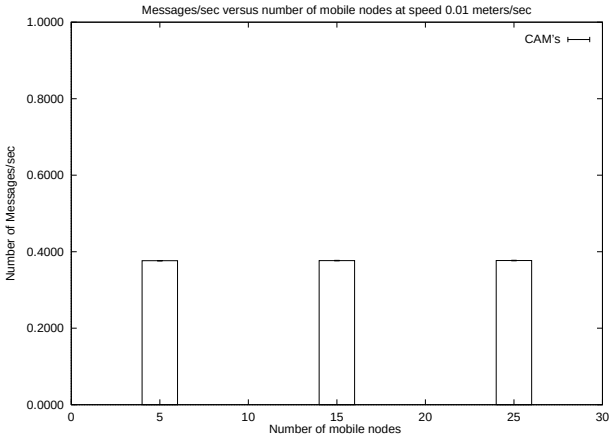
Table 6.2: Events measured over a period of 1900 seconds

in the number of event-driven CAMs and events for very slow speeds, no matter how many nodes move. At 0.1 meters/s the increase is still low, about 1.4% in the case of event-driven CAMs and about 1.3% in the case of events. The last speed is 1.0 meters/sec (3.6 km/hr) which approximately corresponds to the walking speed of a person with a handheld device. Here, we see a 51% of increase in the number of messages as we move from 5 mobile nodes to 25 mobile nodes. The increase in the number of events is about 58%.

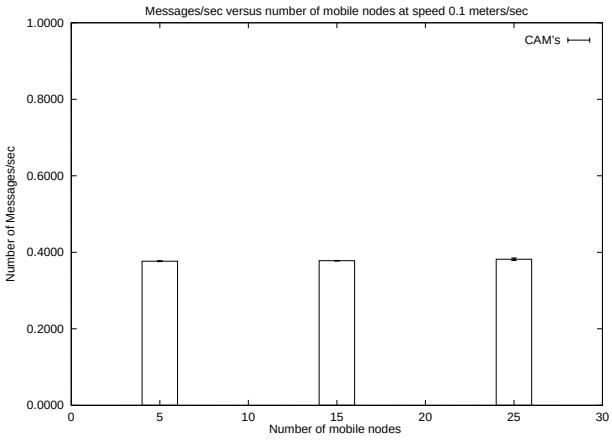
The messages/sec and the events/sec are shown in Figs. 6.15 and 6.16. As can be seen in the graphs the standard deviation is also much higher at the speed 1.0 m/sec. This is because at lower speeds the connectivity does not change much no matter how many nodes move and which nodes move. At higher speeds, the connectivity comes into play. When a node with higher connectivity moves it causes much more exchange of messages than when a node of lower connectivity moves. Hence, we see a wide variation.

6.5 Summary

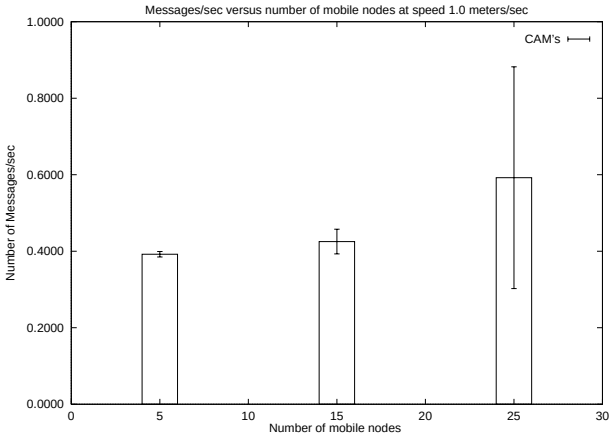
In this chapter, we discussed the results of the simulation experiments of the Code Assignment Algorithm. The simulations were done using the C++ Protocol Toolkit. The results indicate that correct channel assignment using the Code Assignment Algorithm results in approximately 38% increase in throughput. We also see that the number of messages and events to be processed increases linearly as one-hop connectivity increases. This indicates good scalability as the performance of the algorithm is dependent on only local topology and not the global topology of the network. Finally, we have tested the algorithm in a mobile environment and tabulated the messages and events for various speeds.



(a)

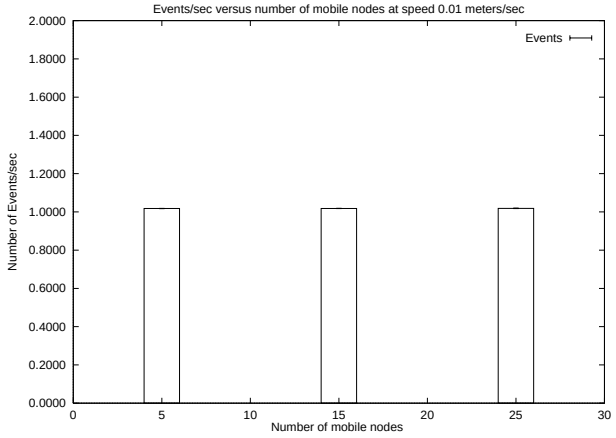


(b)

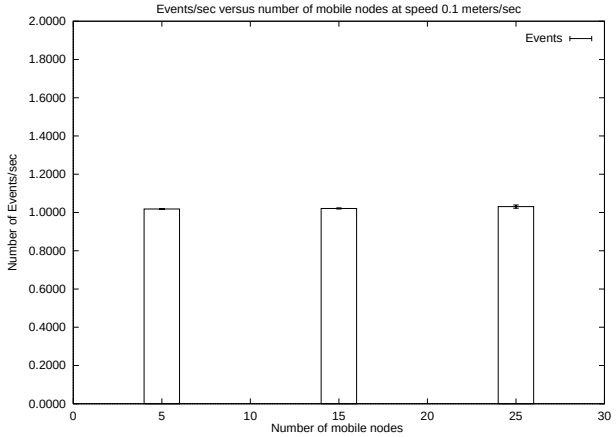


(c)

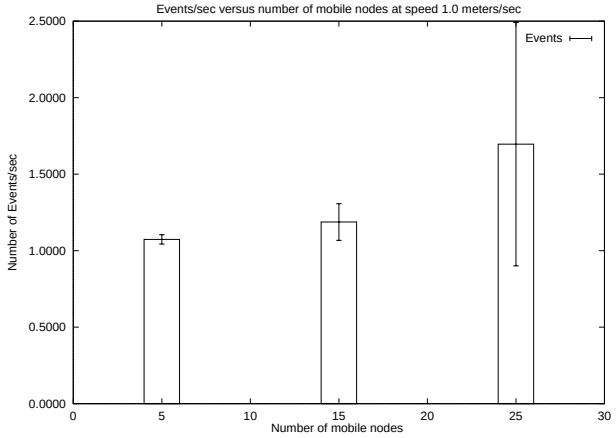
Figure 6.15: Number of event-driven Messages/sec at speeds of (a) 0.01 m/sec (b) 0.1 m/sec (c) 1.0 m/sec



(a)



(b)



(c)

Figure 6.16: Events/sec at speeds of (a) 0.01 m/sec (b) 0.1 m/sec (c) 1.0 m/sec

7. Conclusions

7.0.1 Results

Dynamic channel assignment in multihop radio networks and code assignment in CDMA networks can be mapped onto the two-hop node coloring problem in graph theory which is known to be NP-complete. Since the problem is NP-complete, all polynomial time solutions necessarily have to be heuristic/greedy solutions.

We have presented an algorithm for dynamic channel assignment in multihop packet-radio networks. This algorithm can also be used for code assignment in CDMA networks. This algorithm is completely distributed and does not rely on base stations. Therefore, it is ideal for use in ad-hoc networks. The algorithm is based on using a control channel to obtain information about the transmission codes used by nodes one-hop and two-hops away. Information about the transmission codes is disseminated using the control messages and not tokens or any such mechanism prone to failure.

The algorithm was shown to be correct and its complexity was analyzed [GLAR97]. The storage requirement of the algorithm is very small typically proportional to the square of the degree of the network. The number of messages generated are also proportional to the square of the degree of the network. Simulations of the algorithm were done in CPT. The algorithm was implemented and interfaced to the routing protocol (WIRP) and the MAC protocol (FAMA) implemented in Wireless Internet Gateways (WINGS) [Win, ea97]. We showed that the algorithm results in a substantial increase in throughput. The algorithm scales well and adapts to mobility well. To check this, we measured the number of code assignment messages, the number of events and the time the algorithm takes to converge. The time to converge was shown in simulations to depend on the maximum degree of the network.

7.0.2 Future Work

As the algorithm stands now, in a pathological case a change of code could result in an entire network changing its code. Future work could include looking for ways to limit these changes to previously demarcated areas.

Another research problem would be to look into what kind of end-to-end guarantees can be given in a network where reservation of channels can be done for requested amounts of time. Research in this direction could lead to interesting solutions to making ad-hoc networks capable of supporting integrated services. This research direction includes making the notion of a "code" assignment applicable to scheduling of transmissions over time and frequencies.

References

[Abr70] N. Abramson. The ALOHA System - Another Alternative for Computer Communications. In *Proc. Full Joint Computer Conference*, pages 281–85, 1970.

[Abr73] N. Abramson. Packet switching with satellites. In *Nat. Comput. Conf., AFIPS Conf. Proc.*, volume 42, pages 695–702, Montvale, N.J., 1973.

[BB95] A.A. Bertossi and M.A. Bonuccelli. Code Assignment for Hidden Terminal Interference Avoidance in Multihop Radio Networks. *IEEE/ACM Trans. on Networking*, 3(4):441–449, August 1995.

[BG94] D. Bertsekas and R. Gallager. *Data Networks.* Prentice-Hall Inc., 2nd edition, September 1994.

[CF94] Imrich Chlamtac and Andras Farago. Making Transmission Schedules Immune to Topology Changes in Multi-Hop Packet Radio Networks. *TON*, 2(1):23–29, February 1994.

[CK85] I. Chlamtac and S. Kutten. On Broadcasting in Radio Networks- Problem Analysis and Protocol Design. *IEEE Trans. Computers*, COM-33(12), December 1985.

[CK87] I. Chlamtac and S. Kutten. Tree-Based Broadcasting in Multihop Radio Networks. *IEEE Trans. Computers*, October 1987.

[CS88] I. Cidon and M. Sidi. Distributed Assignment algorithms for Multihop Packet-Radio Networks. *Proc. IEEE Infocom*, pages 1110–1118, 1988.

[ea97] J.J. Garcia-Luna-Aceves et. al. Wireless Internet Gateways (WINGS). In *Proc. IEEE MILCOM 97,*, Monterey, CA, 1997.

[ET90] A. Ephremides and T.V. Truong. Scheduling Broadcasts in Multihop Radio Networks. *IEEE Trans. Computers*, 38(4):456–460, April 1990.

[FG95] C.L. Fullmer and J.J. Garcia-Luna-Aceves. FAMA-PJ:A Channel Access Protocol for Wireless LANs. *Proc. ACM Mobile Computing and Networking*, 1995.

[FGLA97] C. L. Fullmer and J.J. Garcia-Luna-Aceves. Solutions to Hidden Terminal Problems in Wireless Networks. In *Proc. ACM SIGCOMM '97*, Cannes, France, 1997.

[GLAR97] J.J. Garcia-Luna-Aceves and Jyoti Raju. Distributed Assignment of Codes for Multihop Packet-Radio Networks. In *Proc. IEEE MILCOM97*, November 1997.

[HS88] B. Hajek and G. Sasaki. Link Scheduling in Polynomial Time. *IEEE Trans. Inform. Theory*, 34(5):910–917, September 1988.

[Hu93] L. Hu. Distributed Code Assignment for CDMA Packet Radio Networks. *IEEE/ACM Trans. on Networking*, 1(6):668–677, Dec 1993.

[IEE91] IEEE. *P802.11-Unapproved Draft. Wireless LAN Medium Access Control(MAC) and Physical Specifications*, Nov 1991.

[Kar90] P. Karn. MACA - a new channel access method for packet radio. In *ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, pages 134–40, 1990.

[KL73] L. Kleinrock and S. Lam. Packet-switching in a slotted satellite channel. volume 42, pages 703–710, Montvale, N.J., 1973.

[Mak87] T. Makansi. Trasmitter-Oriented Code Assignment for Multihop Radio Networks. *IEEE Trans. Computers*, COM-35(12):1379–1382, Dec 1987.

[MGLA96] S. Murthy and J.J. Garcia-Luna-Aceves. An Efficient Routing Protocol for Wireless Networks. *ACM Mobile Networks and Applications Journal*, 1(2):183–197, Oct 1996.

[Ric] Ricochet Wireless Network. *http://www.ricochet.net/netoverview.html.*

[RKM95] R. Meidan R. Kohno and L.B. Milstein. Spread Spectrum Access Methods for Wireless Communications. *IEEE Communications Magazine*, pages 58–67, January 1995.

[RL93] S. Ramanathan and E. L. Lloyd. Scheduling Algorithms for Multihop Radio Networks. *TON*, 1(2):166–177, April 1993.

[Tan96] A. S. Tanenbaum. *Computer Networks.* Prentice Hall, 3rd edition, 1996.

[TK75a] F. A. Tobagi and L. Kleinrock. Packet Switching in radio channels: Part I - Carrier Sense Multiple-Access Modes and their Throughput-Delay Characteristics. *IEEE Trans. Commun.*, COM-23(12):1400–1416, December 1975.

[TK75b] F. A. Tobagi and L. Kleinrock. Packet Switching in radio channels: Part II - the hidden terminal problem in carrier sense multiple-access modes and the busy-tone solution. *IEEE Trans. Commun.*, COM-23(12):1417–1433, December 1975.

[TK76] F. A. Tobagi and L. Kleinrock. Packet Switching in Radio Channels: Part III - Polling and (Dynamic) Split-Channel Reservation Multiple Access. *IEEE Trans. Commun.*, 24(8), 1976.

[VBZ94] S. Shenker V. Bhargavan, A. Demers and L. Zhang. MACAW: A Media Access Protocol for Wireless LAN's. In *Proc. ACM SIGCOMM '94*, pages 212–25, London, UK, 1994.

[Win] Wireless Internet Gateways. *http://www.cse.ucsc.edu/research/ccrg /projects/wings.html.*